Exercice 1 – jeu de Marienbad

Sur une table sont disposés p tas d'allumettes avec $p \geqslant 1$: le premier tas contient N_1 allumettes, le second N_2, et ainsi de suite jusqu'au p-ième tas contenant N_p allumettes. Initialement, les entiers $N_1, \dots, N_p$ sont non tous nuls.
Le jeu se joue à deux. Le premier joueur retire un nombre strictement positif d'allumettes d'un tas, puis le second fait de même, puis le premier, etc... Le jeu s'arrête lorsque toutes les allumettes ont été retirées, le joueur gagnant étant celui qui a pris la ou les dernières allumettes présentes sur la table.
Dans tout l'exercice, les fonctions demandées sont à rédiger en Python.

Partie 1 - manipulation d'entiers en base 2

On rappelle que tout entier naturel n se décompose de manière unique comme somme de puissances de 2 deux à deux distinctes. Cette écriture se note

$$n = \sum_{k=0}^{+\infty} n_k 2^k \quad \text{avec} \quad n_k \in \{0,1\}$$

La suite $(n_k)_{k\in\mathbb{N}}$, qui est nulle à partir d'un certain rang, s'appelle la décomposition en base 2 de n. Elle est composée des restes des divisions euclidiennes successives de n par 2. Par exemple, en divisant $n = 11$ par 2 , on obtient les quotients successifs $5, 2, 1, 0, 0, \dots$ et les restes $n_0 = 1, n_1 = 1, n_2 = 0, n_3 = 1, n_4 = n_5 = \dots = 0$ et ainsi

$$11 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3.$$

1. Une fonction `tobin` prend en argument un entier n et renvoie la liste $[n_0, \dots, n_p]$ d'éléments de $\{0,1\}$, de plus petite longueur, telle que

$$n = \sum_{k=0}^{p} n_k 2^k \quad \text{avec} \quad n_k \in \{0,1\} \qquad (\star)$$

Par exemple `tobin(11)` renverra `[1,1,0,1]`
Compléter cette fonction :

```
def tobin(n):
    L = []
    if n == 0:
        return [0]
    q = n
    while q > 0:
        .....
        .....
    return L
```

2. À l'inverse, écrire une fonction `frombin` prenant en argument une liste $[n_0, \dots, n_p]$ et renvoyant l'entier n défini par la formule $(\star)$. On veillera à minimiser le nombre de multiplications (attention, $a^{**}k$ se fait en k multiplications).

On définit l'opération $\oplus$ de $\{0,1\}^2$ dans $\{0,1\}$ par

$$\forall x, y \in \{0,1\}, \quad x \oplus y = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases}$$

3. Définir la fonction `xorb` prenant en argument deux entiers a et b et renvoyant $a \oplus b$ si $a, b \in \{0,1\}$. On utilisera une assertion pour vérifier cette dernière condition (entier et valant 0 ou 1).

Étant donnés deux entiers n et m décomposés en base 2

$$n = \sum_{k=0}^{\infty} n_k 2^k \quad \text{et} \quad m = \sum_{k=0}^{+\infty} m_k 2^k$$

on pose par abus de notation

$$n \oplus m = \sum_{k=0}^{\infty} (n_k \oplus m_k)\, 2^k.$$

On admet que cette opération est commutative et associative ($x \oplus y = y \oplus x$ et $(x \oplus y) \oplus z = x \oplus (y \oplus z)$).

4. Calculer $1 \oplus 3 \oplus 5 \oplus 7$ et $1 \oplus 3 \oplus 4 \oplus 7$.

5. Écrire une fonction `complete` prenant en arguments deux listes de 0 et de 1 et qui renvoie ces deux listes en complétant la liste la plus courte avec des 0 à la fin afin d'obtenir la même longueur. Par exemple :

```
>>> complete([1,0,1], [1,0,0,0,1])
([1, 0, 1, 0, 0], [1, 0, 0, 0, 1])
```

6. Écrire une fonction `xor` prenant en argument deux entiers naturels n et m quelconques et renvoyant $n \oplus m$ (on prendra garde au fait que deux listes renvoyées par la fonction `tobin` ne sont pas nécessairement de même longueur).

Partie 2 - élaboration d'une stratégie gagnante

Une configuration du jeu est entièrement caractérisée par la donnée du nombre d'allumettes de chacun des tas. On code donc une configuration par un p-uplet d'entiers naturels $(\mathrm{N}_1, \ldots, \mathrm{N}_p)$.
On dit qu'une configuration est favorable lorsque $\mathrm{N}_1 \oplus \mathrm{N}_2 \oplus \cdots \oplus \mathrm{N}_p$ est **non nul**. Sinon, on dit qu'elle est défavorable. On note A et B les deux joueurs.

7. La configuration $(1,3,5,7)$ est-elle favorable ? Si c'est le cas, déterminer tous les coups que l'on peut jouer qui placent le jeu dans une configuration défavorable. Même question avec $(1,3,4,7)$.

Soient $x_1, \ldots, x_p$ des entiers naturels et $\mathrm{X} = x_1 \oplus x_2 \oplus \cdots \oplus x_p$.

8. On suppose que $\mathrm{X} = 0$. Soit $i_0 \in [\![1; p]\!]$ et $z \in \mathbb{N}$ différent de x_{i_0}. On pose $y_i = x_i$ si $i \neq i_0$ et $y_{i_0} = z$. Justifier que

$$y_1 \oplus y_2 \oplus \cdots \oplus y_p \neq 0$$

9. On suppose que $\mathrm{X} \neq 0$. Justifier l'existence d'un entier $i_0 \in [\![1; p]\!]$ et de $z \in \mathbb{N}$ avec $z < x_{i_0}$ de sorte qu'avec les mêmes notations qu'à la question précédente,

$$y_1 \oplus y_2 \oplus \cdots \oplus y_p = 0$$

(Si X se décompose en binaire $\mathrm{X} = \sum_{k=0}^{+\infty} \mathrm{X}_k 2^k$, on pourra introduire $k_0 = \max\{k \in \mathbb{N}, \mathrm{X}_k \neq 0\}$.)
Montrer que pour un tel i_0, on a nécessairement $z = \mathrm{X} \oplus x_{i_0}$.

10. Le joueur A récupère la main et le jeu est dans une configuration favorable. A a-t-il une stratégie gagnante ? Si oui laquelle ? Si non, peut-il encore gagner ? En faisant quoi ?

11. Le joueur A récupère la main et le jeu est dans une configuration défavorable. A a-t-il une stratégie gagnante ? Si oui laquelle ? Si non, peut-il encore gagner ? En faisant quoi ?

12. Écrire une fonction `coup_suivant` qui prend en entrée une liste N de taille p contenant les valeurs de $\mathrm{N}_1, \ldots, \mathrm{N}_p$, et modifie la liste de sorte que celle-ci contienne la configuration obtenue après avoir joué un des coups correspondants à la meilleure stratégie détaillée lors des deux questions précédentes.

Exercice 2 – base de données

On considère la base de données de gestion des séries dans une plateforme de diffusion dont le schéma relationnel est le suivant :

```
serie(id_serie INT, titre_serie TEXT, annee_crea_serie INT, annee_fin_serie INT, pays_serie TEXT)
saison(id_sais INT, num_sais INT, annee_lance_sais INT, id_serie INT)
episode(id_epis INT, num_epis INT, titre_epis TEXT, duree_epis INT, id_sais INT)
plateforme(id_pf INT, nom_pf TEXT, pays_siege_social_pf TEXT, nom_creat_pf TEXT)
diffusion(id_sais INT, id_pf INT, pays_diff TEXT, annee_diff INT)
```

Cela signifie qu'il y a cinq tables :

- `serie` : décrite par un identifiant `id_serie` de type entier, un titre `titre_serie` de type texte, une année de création `annee_crea_serie`, une année de fin `annee_fin_serie`, et un pays d'origine `pays_serie` ;
- `saison` : décrite par un identifiant `id_sais`, un numéro de saison `num_sais`, une année de lancement `annee_lance_sais`, un identifiant de série `id_serie` ;
- `episode` : décrite par un identifiant `id_epis`, un numéro d'épisode `num_epis`, un titre `titre_epis`, une durée (en minutes) `duree_epis` et la saison à laquelle il appartient `id_sais` ;
- `plateforme` : décrite par un identifiant `id_pf`, un nom `nom_pf`, le pays d'origine du siège social `pays_siege_social_pf`, le nom du créateur de la plateforme `nom_creat_pf` ;
- `diffusion` : décrite par une saison (d'une série) `id_sais`, une plateforme `id_pf`, un pays `pays_diff`, une année `annee_diff`.

Les différents pays sont représentés par leur code : USA pour les Etats-Unis d'Amérique, FR pour la France, D pour l'Allemagne, GB pour la Grande-Bretagne...

1. Donner une clé primaire pour chacune des tables.

Pour les questions suivantes, écrire une requête permettant de répondre à la question.

2. Afficher les informations sur les séries françaises crées en 2022.

3. Afficher le nombre de séries françaises crées en 2022.

4. Afficher les titres des épisodes de la saison 1 de la série d'identifiant 123 en les classant par ordre croissant de numéros.

5. Afficher, pour la série GAME OF THRONES, les numéros de saison classés par ordre croissant ainsi que la durée totale en minutes pour chaque saison.

6. Afficher les noms des deux séries les plus longues (en minutes).

7. Afficher les titres des séries dont au moins un épisode a été diffusé en 2023 aux USA.

8. Afficher les titres des séries dont au moins un épisode a été diffusé en 2023 aux USA, mais aucun en France la même année.

9. Afficher les titres des séries dont au moins deux saisons ont été diffusées (même partiellement) en France en 2023 par la plateforme NFX, ainsi que le nombre exact de saisons diffusées.

Exercice 3 – algorithme kNN

Dans la série de livres Harry Potter, les élèves, dès leur arrivée à l'école de Poudlard, sont répartis dans l'une des différentes maisons (Gryffondor, Serpentard, Serdaigle et Poufsouffle). Cette affectation est déterminée par le Choixpeau. Loin de faire appel à de la magie pour affecter les élèves, le Choixpeau semble être une intelligence artificielle utilisant l'algorithme des k plus proches voisins (kNN) pour affecter les élèves.
Afin de procéder à cette affectation, on dispose d'un tableau `eleves` (liste de liste) répertoriant des caractéristiques (courage, loyauté, sagesse, malice) et l'affectation d'élèves antérieurs. Par exemple :

```
>>> eleves[0]
['Adrian', 9, 4, 7, 10, 'Serpentard']
>>> eleves[1]
['Andrew', 9, 3, 4, 7, 'Gryffondor']
```

L'idée est, étant donnée une liste comportant un nom et les quatre entiers correspondant aux caractéristiques, d'obtenir une proposition d'affectation dans une maison.

1. Écrire une fonction `distance` d'arguments deux listes de quatre entiers et renvoyant le carré de la distance euclidienne entre les deux vecteurs de $\mathbb{R}^4$ correspondants. Par exemple :

```
>>> distance([4, 5, 6, 7], [8, 5, 4, 6])
21
```

2. Écrire une fonction `ToutesDistances` qui prend en argument une liste correspondant à un nouvel élève (de longueur 5, le nom suivi des 4 entiers) et qui renvoie une liste de couples de la forme `(entier, nom)` correspond aux distances avec chacun des élèves déjà répertoriés, suivi de l'affectation correspondante.

```
>>> ToutesDistances(['Hermione', 8, 6, 6, 6])
[(22, 'Serpentard'), (15, 'Gryffondor'), ...]
```

   (les points de suspension indiquent que ce résultat n'est pas complet).

3. On suppose que l'on dispose d'une fonction `tri` permettant de trier une liste de couples du type précédent par ordre croissant du premier argument.

   Écrire une fonction `MaisonMajoritaire` qui prend en argument une liste correspondant à un nouvel élève (de longueur 5, le nom suivi des 4 entiers) et qui renvoie la maison la plus fréquente parmi les 11 élèves les plus proches (on ne considère pas le cas d'égalité de distances, ni le fait qu'il puisse y avoir plusieurs maisons "majoritaires", on souhaite juste une possibilité).